Particle Filter

Shubhankar Kulkarni

November 24, 2022

1 Introduction

The report discusses the application of the **Particle Filter** for tracking problems where the probability distribution of the states to be tracked does not necessarily follow a Gaussian probability distribution. The nonlinear state transitions follow a multi-modal distribution which renders the Kalman and the Extended Kalman filter to be of no help.

For such problems, the approach of defining an equation that will describe the distribution modelling is abandoned. The focus is instead made to approximate the distribution using a combination of state values with some associated weights to them. The concepts of state transition equation, observation equation, dynamic and measurement noises from the Kalman Filters still hold for describing the transition of the subsequent states and weights from the previous ones. Each weight is a metric to represent the probabnility of it's corresponding state value representing the actional state provided the measurement.

However, we are not provided with any such calculation of the fidelity representation of a given state with the actual value. To mitigate this, sampling techniques are applied that give us the expected value of the state at that stage by factoring in all of the discrete point state values.

2 Methods

The Particle filter follows the similar predict and update model of the Kalman Filter without the cimputation of a Kalman Gain factor. The weight update happens using the Recursive Bayesian estimate which basically gives the fidelity degree of a state to represent the true value based upon the measurement recorded

$$p(x_{0:t+1}|y_{0:t+1}) \tag{1}$$

this is based upon the fiedlity estimation of the previous stage state

$$p(x_{0:t}|y_{0:t}) \tag{2}$$

.

2.1 Distribution Approximation

The main difference is the usage of Monte Carlo principle to approximate the multi-modal distribution using a bunch of particles. This can be represented as:

$$\chi = \{x^{(m)}, w^{(m)}\}_{m=1}^{M} \tag{3}$$

where $x^{(m)}$ represents the state and $w^{(m)}$ represents the weight of a single sample (m). A figurative representation of the approximation model can be seen below. An arbitrary number of particles can be assigned to represent the distribution, each with their own estimation of state value and a weight attached based upon the subsequent sampling calculations.

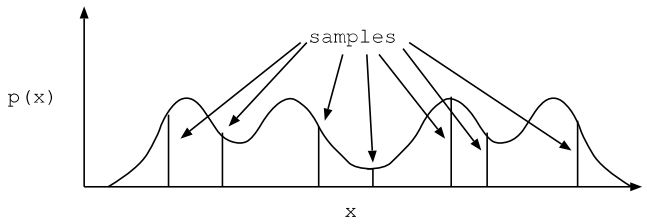

Figure 1: Monte Carlo approximation of a non-Gaussian distribution.

2.2 Sampling technique and Weight Update

The technique to append a specific weight to a specific particle since the fidelity metric based upon the measurement $p(x|y)$ is not known. Importance Sampling provides a methodology for us to work on this problem by defining the expected value of $x|y$

$$E_p[x] = \int x \frac{p(x|y)}{q(x|y)} q(x|y) dx \tag{4}$$

It is the value of all possible states x across the probability of each of those states given the measurement. Another distribution called as the proposal distribution is used in the equation which is a known distribution which could be anything from Gaussian to other multi-modal distributions. The weights are calculated as

$$w_t^{(m)} = \frac{p(x_{0:t}^{(m)}|y_{0:t})}{q(x_{0:t}^{(m)}|y_{0:t})} \tag{5}$$

The iteration is done for t stages conisdering that we can obtain the states and measurements at each of the stages for each of the m particles. Here m is the index of the particle and $x_t^{(m)}$ is the state vector of a particle at an arbitrary stage t and y_t is the common sensor measurement vector at that stage comparing with which the particle weights are updated. Plugging in the recursive Bayesian estimation, we get the corresponding state transition equation, the observation equation and the previous estimate of state in the context of the iterative framework.

$$\tilde{w}_t^{(m)} = \frac{p(x_t^{(m)}|x_{t-1}^{(m)}) p(y_t|x_t^{(m)})}{q(x_t^{(m)}|x_{0:t-1}^{(m)}, y_{0:t})} w_{t-1}^{(m)} \tag{6}$$

The proposal distribution $q(x_t^{(m)}|x_{0:t-1}^{(m)}, y_{0:t})$ can be any arbitrary distribution. For the sake of computation simplicity, we assume it to be equal to the $p(x_t^{(m)}|x_{t-1}^{(m)}$. This effectively renders the weight update to be

$$\tilde{w}_t^{(m)} = p(y_t|x_t^{(m)}) * w_{t-1}^{(m)} \tag{7}$$

Here $w_t^{(m)}$ corresponds to the m^{th} particle weight in the t stage. The weights of each particle would then be needed to be normalized so that the sum of weights remains 1:

$$w_t^{(m)} = \frac{\tilde{w}_t^{(m)}}{\sum_{m=1}^{M} \tilde{w}_t^{(m)}} \tag{8}$$

2.3 Problem statement & Particle Filter Algorithm

The problem represents a system in a 1D motion. The position of the system is a sinusoidal motion pattern when evaluated with respect to time. Two sensors detecting magnetic field strength are placed which results in the field strength being a sum of the distances of the moving body from the two magnets. The first task in approaching the problem is building the state space:

The terminologies used in the subsequent lines is highlighted here:

- X_t, a set of state variables
- A_t, the set of dynamic noises
- f(), the state transition equation
- Y_t, the set of measurements
- N_t, the set of measurement noises
- g(), the observation equation

The states for an arbitrary t stage for each of the m particles undergo the state transition wherein they depend on the state at the $t-1$ stage for the particular particle. This logic flow is equivalent to running a seperate Kalman filter for each of the particles. We consider two state variables:

$$X_t = \begin{bmatrix} x_t \\ \dot{x}_t \end{bmatrix} \tag{9}$$

The state transition equation is as follows:

$$f(x_t, a_t) = \begin{bmatrix} x_{t+1} = x_t + \dot{x}_t T \\ \dot{x}_{t+1} = \begin{cases} 2 & \text{if } x_t < -20 \\ \dot{x}_t + |a_t| & \text{if } -20 \leq x_t < 0 \\ \dot{x}_t - |a_t| & \text{if } 0 \leq x_t \leq 20 \\ -2 & \text{if } x_t > 20 \end{cases} \end{bmatrix} \tag{10}$$

The piecewise function defining next velocity state associates a random amount a_t to the current velocity which has a gaussian distribution of zero mean $N(0, \sigma_a^2)$ with a standard deviation of $sigma_a = 2^{-4} = 0.0625$. The position constraints are $-20 < x_t < 20$.

An important note to make is that the weight of each particle is equal to $1/M$ at the start where M is the number of particles.

The observation equation can be defined by the magnets measuring the total magnetic strength represented as:

$$Y_t = \begin{bmatrix} y_t \end{bmatrix} \tag{11}$$

Two magnets are placed at $x_{m1} = -10$ and $x_{m2} = 10$. The observation equation for this model is:

$$g(x_t, n_t) = \left[y_t = \frac{1}{\sqrt{2\pi}\sigma_m} \exp(\frac{-(x_t - x_{m1})^2}{2\sigma_m^2}) + \frac{1}{\sqrt{2\pi}\sigma_m} \exp(\frac{-(x_t - x_{m2})^2}{2\sigma_m^2}) + n_t \right] \quad (12)$$

where n_t is a random sample drawn from $N(0, \sigma_n^2)$ representing measurement noise. The value $\sigma_m = 4.0$. The data was generated using a value of $\sigma_n = 2^{-8} = 0.003906$.

The weight update and normalization for each particle happen as per equations 7 and 8. But for calculating $p(y_t|x_t^{(m)})$ for that particle, we need to do the following-

We first need to perform the task of calculating an ideal measurement for each of the particles assuming there is zero measurement noise. It can be represented as follows:

$$g(x_t^{(m)}, 0) = \left[y_t^{(m)} = \frac{1}{\sqrt{2\pi}\sigma_m} \exp(\frac{-(x_t^{(m)} - x_{m1})^2}{2\sigma_m^2}) + \frac{1}{\sqrt{2\pi}\sigma_m} \exp(\frac{-(x_t^{(m)} - x_{m2})^2}{2\sigma_m^2}) \right] \quad (13)$$

This ideal measurement can then be compared against the actual measurement in the model of the measurement noise for each of the particles as follows:

$$p(y_t|x_t^{(m)}) = \frac{1}{\sqrt{2\pi}\sigma_n} \exp(\frac{-(y_t^{(m)} - y_t)^2}{2\sigma_n^2}) \quad (14)$$

This now can be used to calculated the updated and normalized weight for that particle. After this, the desired output for that stage involving all of the particles is computed, such as the expected value (mean):

$$E[x_t] \approx \sum_{m=1}^{M} x_t^{(m)} \cdot w_t^{(m)} \quad (15)$$

It can also be anythign other than the expected value (mean) such as the local maxima.

The general phenomenon is that the particles which follow the trend of the measurement data get assigned an increasing weight as the iterations for time stages proceed. This means that the weights of the particles that stray away get reduced and after a while they become insignificant with zero weights. This continual reduction in the number of high weight particles would mean that after a while, there would be lesser number of particles which would not represent the probability distribution correctly. This will defeat the primary purpose of using particle filters to respresent a multi-modal distribution problem.

To avoid this loss of representation, we resort to resampling of the particles wherein an observation is taken and the weight of each particle is updated according to it's prediction fidelity towards the observation. Renormalizing the

weights results in the sum being 1 and thereby we once again have a probability distribution. This task is computationally intensive and hence needs to be triggered post certain conditions being met. To determine if resampling is necessary, we calculate the coefficient of variation:

$$\mathrm{CV} = \frac{\mathrm{VAR}(w^{(m)})}{E^2[w^{(m)}]} = \frac{\frac{1}{M}\sum_{m=1}^{M}\left(w^{(m)} - \frac{1}{M}\sum_{m=1}^{M} w^{(m)}\right)^2}{\left(\frac{1}{M}\sum_{m=1}^{M} w^{(m)}\right)^2} = \frac{1}{M}\sum_{m=1}^{M}(M\cdot w^{(m)} - 1)^2 \tag{16}$$

The weights used here are after updation of weights for that iteration. The effective sample size ESS is then calculated from the CV and number of particles M as follows:

$$\mathrm{ESS} = \frac{M}{1 + \mathrm{CV}} \tag{17}$$

If the ESS is less than a certain percent of the total number of particles, we can decipher that the particles having an appreciable weight has fallen down to that much percent. This provides us a criteria of wehther to resample or not. For our purposes, we have set 50% as the threshold for resampling of the particles. In the resampling algorithm, the list of particles is populated again during which the particles having lower weights have a lesser chance of being copied and particles with more weights have more copies of them. This results in the new list of particles trending in the direction of the previously higher weighted particles ensuring that the filter follows the trend of the measurement.

3 Observations

3.1 1D position tracking

At the start, all of the particles are initialized to have the same position and velocity with a uniform weight. Dynamic noises with a zero mean gaussian distribution of standard deviation 0.0625 are used in the piece-wise state transition equations to calculate particle velocities which also leads to them assuming different positions as iterations progress.

The weights of the particles get updated as per the comparison of the ideal measurement using particle's position with the actual measurement by the sensors at that stage wherein a random measurement noise with a zero-mean gaussian distribution with 0.003906 standard deviation is used. The measurement is the field strength which is a different physical quantity than the position but gives an indication of it. Consequently, the particles which resemble the actual measurement better get increasingly more weight assigned to them than the ones which stray away and after a while when the particles stray away enough and very less particles truly represent the distribution, the resampling condition is triggered. This leads to all M particles being newly initialized in trend to the previously higher weighted particles and all of them get assigned uniform weights again.

Due to the problem being symmetric in nature, the particle filter keeps on fluctuating with the actual position measurement in the phase of $0°$ and $180°$ respectively since there is no state corresponding to direction. It can be observed that the filter takes time to converge to the real position values but once it latches onto the trend, it represents the sinusoidal behavior of position with respect to time quite accurately which the Kalman and Extended Kalman filters would not have been able to due to the distribution not being a zero-mean gaussian.

Given below is the plot of 180° phase variation in which we can see the mismatch in the estimated filter output and the actual position data at the start whereas the filter follows the data pretty closely after it but with a constant opposite phase.

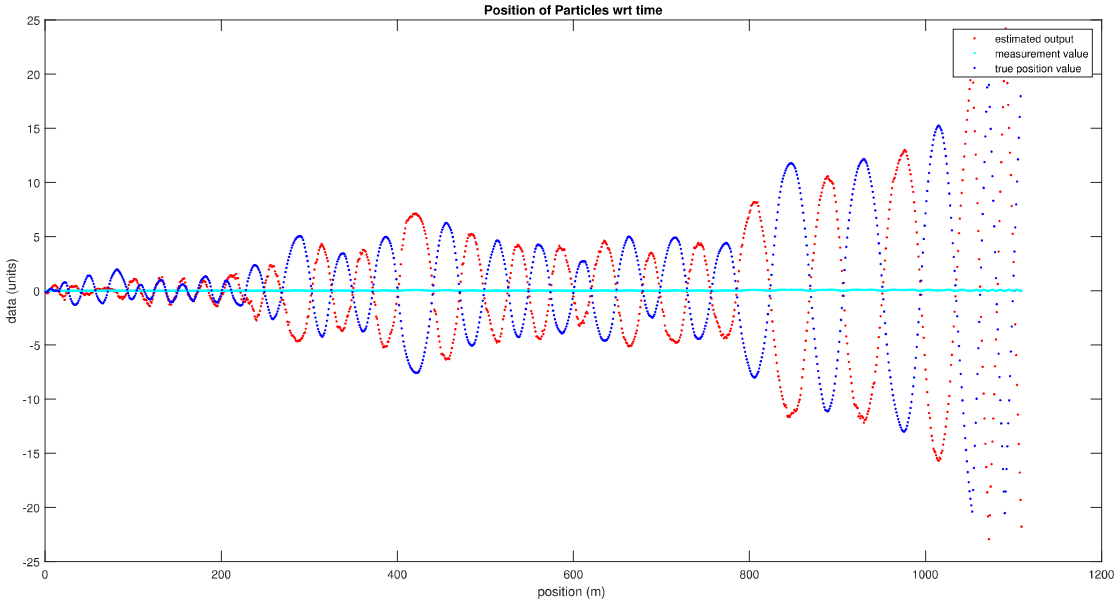

Figure 2: Filter output and actual measurement data with opposite phase with time

For the above figure, the actual position data is of lower amplitudes at the initial stages and it seems to keep on growing as time progresses till it keeps on saturating at both 20 and -20 limits of position. When the actual position is lesser at the start, the resampling condition is triggered after many iterations indicating that the change in the nubmer of particles truly representing the distribution is gradual but as the value increases, the resampling condition is triggered more frequently and at the peak of variation (i.e. at the end), the condition gets triggered after every 3 or 4 iterations indicating that the particles stray much more quickly with respect to the actual measurement once the change in true position is drastic.

Given below is the plot of 0° phase variation in which we can see the matching in the estimated filter output and the actual position data at the start whereas the filter follows the data pretty closely after it but with almost coinciding values.

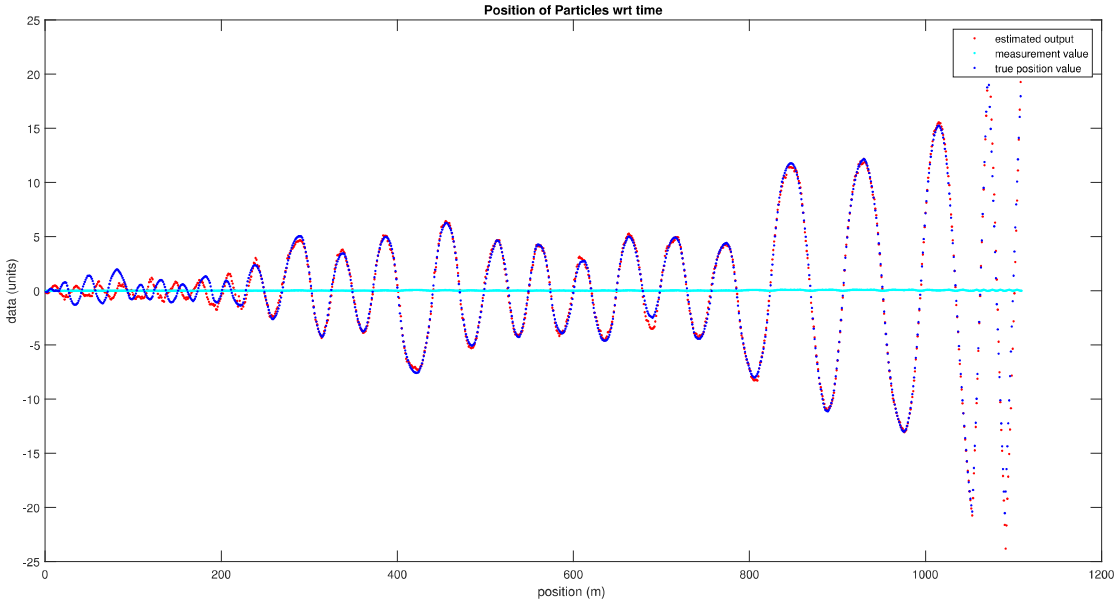

Figure 3: Filter output and actual measurement data with zero phase with time

For the above figure, the estimated filter output seems to take time to converge to the actual data values and the resampling occurs less frequently when the true position value change is relatively small. The estimated output matches the actual value and resampling occurs more frequently as the time progresses in an increasing order till the end.

3.2 Multi-modal Distribution Analysis

The plot below shows the position versus weight plot indicating the distribution when the resampling condition isn't called. The number of particles has been increased to 1000 for better visualization of the particle distribution curve. The most evident thing is that the distribution is not a Gaussian one and hence the compoutationally expensive Particle Filter's use is justified over the lesser intensive Gaussian filters. It can be seen that the distribution is much more discernible compared to the plot below for when the resampling condition is triggered. Most of the particles have their weights between -1 to -2 and lesser values corresponding to 0. This indicates that the particles haven't strayed far enough from the actual measurement that most of their weights become insignificant in the calculation of the estimated value.

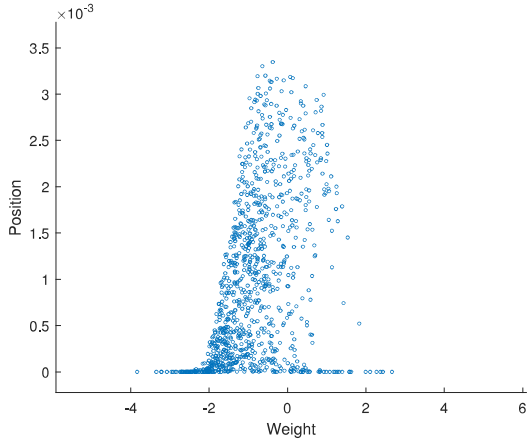

Figure 4: Position versus Weight of each particle indicating probability distribution

Given below is the plot of probability distribution of the particles for a particular time instance. The distribution is not a Gaussian one as seen. The plot is just before the resampling condition is triggered and we can see that a relatively high number of particles have their weights between -0.5 and 0 indicating that they are of lesser significance on the estimated output. Post the resampling, the weight distribtion is uniform because the particles have been re-initialized with $1/M$ weights.

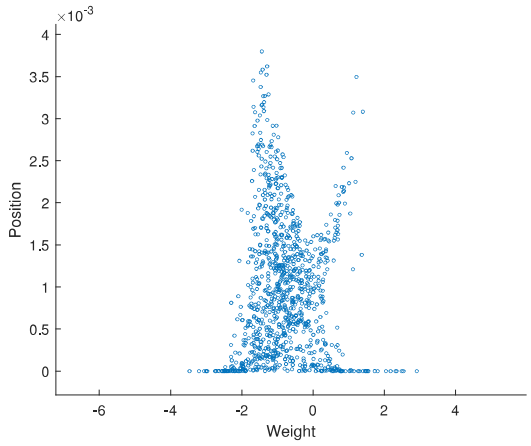

Figure 5: Position versus Weight of each particle indicating probability distribution

4 Results and conclusion

The conclusion that can be drawn from the plots is that the Particle Filter does a good enough job with the given dynamic an measurement noise distributions in calculating the position variation with respect to time. Resampling ensures that the strayed distribution is re-initialized in the areas of previously higher weights and this is the method which particle filter uses to latch onto a multi-modal distribution system which is infeasible by the Gaussian filters.

5 Appendix

5.1 Code: 1D Position Tracking Particle Filter

```
clc;
clear;
clf;
urlwrite("https://cecas.clemson.edu/ahoover/ece854/labs/magnets-data.txt",...
    "measurement_Q1.txt")
%% Populating true values and measurement data
inp_filename = "measurement_Q1.txt"
fid = fopen(inp_filename); % open the file
iCtr = 0;
y_q1=[];
true_x_q1=[];
true_xdot_q1=[];
```

```
while ~feof(fid) % loop over the following until the end of the file is reached.
  line = fgets(fid); % read in one line
  data1 = strsplit(line);
  true_x_q1(end+1) = str2double(data1(1));
  true_xdot_q1(end+1) = str2double(data1(2));
  y_q1(end+1) = str2double(data1(3));
end
true_x_q1=(true_x_q1)';
true_xdot_q1 = (true_xdot_q1)';
y_q1=(y_q1)';
y_q1 = y_q1;
fignum = 1;

time_q1=(1:1:length(y_q1));
time_q1 = time_q1';
%% Particle Filter Logic
%%number of particles for PF
M = 1000;
Est_pos_arr = [];
%assuming a m*4 particle state array with columns corresponding to x_prev,
%xdot_prev, x_curr, xdot_curr
PF_states = zeros(M,4);

%assuming a m*2 particle weight array with columns corresponding to w_prev, w_curr
PF_weights = zeros(M,2);

%Initially assuming the initial x_prev and xdot_prev values for all particles
PF_states(:,1) = true_x_q1(1);
PF_states(:,2) = true_xdot_q1(1);

%Initially assuming a uniform 1/M weight for all particles
PF_weights(:,1)=1/M;

%loop the measurement array
for j=1:1:length(y_q1)
    %Calling the state transition equation for every particle
    for i=1:1:M
        [x_next, xdot_next] = state_transition(PF_states(i,1), ...
        PF_states(i,2), time_q1(2)-time_q1(1));

        PF_states(i,3) = x_next;
        PF_states(i,4) = xdot_next;
    end

    %computing the ideal measurement for each particle
    %comparing the ideal measurement for every particle with the sensor reading
```

```
%for that instant -> p(yt|x(m)t)
%weight update for each of the particles
weight_sum = 0;
for i =1:1:M
    PF_weights(i,2) = update_weight(PF_states(i,3),y_q1(j),PF_weights(i,1));
    %weight_sum = weight_sum + PF_weights(i,2);
end
weight_sum = sum(PF_weights(:,2));
%normalize the weight
for i =1:1:M
    PF_weights(i,2) = PF_weights(i,2)/weight_sum;
end

%compute the expected value at that stage
Exp_val = 0;
Exp_val = PF_weights(:,2).*PF_states(:,3);
%Exp_val = (Exp_val) / M;
Exp_val = sum(Exp_val);
Est_pos_arr(end+1)=Exp_val;

%calculating effective sample size ESS = M/(1 + CV)
CV = 0;
for i =1:1:M
    CV = CV + (M*PF_weights(i,2)-1)^2;
end
CV = CV/M;
ESS = M/(1+CV);

%checking if resampling is necessary if (ESS < 0.5*M)
if(ESS < 0.5*M)
    scatter(PF_states(:,1),PF_weights(:,1),5);
    xlim([-6 6])
    ylim([0 0.004])
    xlabel("Weight")
    ylabel("Position")

    fprintf("Resampling necessary! at iteration %d\n",j);
    [PF_states(:,3),PF_states(:,4),PF_weights(:,2)] = resample(PF_weights(1:end,2),...
    PF_states(1:end,3),PF_states(1:end,4),M);
end

scatter(PF_states(:,1),PF_weights(:,1),5);
xlim([-6 6])
ylim([0 0.004])
xlabel("Weight")
ylabel("Position")
```

```
    %substituting the current states and weights into previous
    PF_weights(:,1) = PF_weights(:,2);
    PF_states(:,1) = PF_states(:,3);
    PF_states(:,2) = PF_states(:,4);
end
%%  Plotting the estimated value, measurement and true state over the course of time
    clf;
    figure(fignum)
    plot(time_q1, Est_pos_arr',.r')
    hold on;
    plot(time_q1, y_q1,'.c')
    hold on;
    plot(time_q1, true_x_q1,'.b')
    ylabel('data (units)');
    xlabel('position (m) ');
    hold on;
    title('Position of Particles wrt time');
    legend('estimated output','measurement value','true position value',...
        'location','northeast');

%% Functions
    function [x_next, xdot_next] = state_transition(x_curr, xdot_curr, T)

        sig_a = 0.0625;
        x_next = x_curr + xdot_curr*T;

       if x_curr < -20
           xdot_next = 2;
       elseif (-20 <= x_curr) && (x_curr < 0)
           xdot_next = xdot_curr + abs(normrnd(0,sig_a));

       elseif (0 <= x_curr) && (x_curr <= 20)
           xdot_next = xdot_curr - abs(normrnd(0,sig_a));
       else
           xdot_next = -2;
       end
    end

    function [w_next] = update_weight(x, y, w_prev)
        sigm = 4.0;
        sign =  0.003906;
        xm1 = -10;
        xm2 = 10;

        %%calculating yt^m
```

```
    y_m = (1/(sqrt(2*pi)*sigm))*exp(-(x-xm1)^(2)/(2*sigm*sigm)) ...
        + (1/(sqrt(2*pi)*sigm))*exp(-(x-xm2)^(2)/(2*sigm*sigm));

    %computing p(yt|x(m)t)
    probab = (1/(sqrt(2*pi)*sign))*exp(-(y_m-y)^(2)/(2*sign*sign));

    %computing weight updation
    w_next = w_prev*probab;
end

function [New_pos,New_vel,NewW] = resample(PF_weights,PF_states_pos,PF_states_vel,M)
        Q = cumsum(PF_weights);
        t = rand(M+1,1);
        T = sort(t);
        T(M+1) = 1.0;
        i=1;
        j=1;
        while (i<=M)
            if T(i) < Q(j)
                Index(i)=j;
                i=i+1;
            else
                j=j+1;
            end
        end

        for i=1:1:M
            New_pos(i)=PF_states_pos(Index(i));
            New_vel(i)=PF_states_vel(Index(i));
            NewW(i)=1/M;
        end
end
```